Optimizing the cloud? Don't train models. Build oracles!

Tiemo Bang, Conor Power, Siavash Ameli, Natacha Crooks, Joseph M. Hellerstein
University of California, Berkeley
tbang@berkeley.edu

ABSTRACT

We propose *cloud oracles* as an alternative to machine learning for online optimization of cloud configurations. Our cloud oracle approach guarantees complete accuracy and explainability of decisions for problems that can be formulated as parametric convex optimizations. We give experimental evidence of this technique's efficacy and share a vision of research directions for expanding its applicability.

1 INTRODUCTION

The cloud offers the promise of an infinitely flexible, elastic, and scalable black box. In practice, to keep costs low, developers must intelligently choose how to deploy their applications between hundreds of storage tiers, hardware types, and regions. This is exacerbated by the move to multi-cloud pipelines and the proliferation of cloud providers [11, 27, 52]. Finding the optimal configuration for a system is far from straightforward. This challenge is further compounded by the fact that one must find this configuration not just once, but multiple times as workload changes, which happens frequently at scale.

Existing solutions cluster around two extremes. On the one hand, Integer Linear Programming (ILP) solvers offer provably correct solutions but at prohibitively high computational cost. For instance, our benchmarks highlighted in Figure 1 show that optimizing the storage location of a single object takes >16 minutes at the current cloud-scale (>100 data centers) [8, 13, 14, 44]. On the other hand are approximation solutions, currently dominated by machine learning techniques. These trade some accuracy for fast execution but suffer from operational challenges of training pipelines and a lack of explainability that hinders deployment in production settings [56, 57]. This accuracy trade-off is also an expensive one at cloud scales where companies are paying hundreds of millions in cloud bills annually. Recent work on ML-based autotuning for cloud databases [54] showed performance variations in the 10s of percents (millions of dollars of waste, at the scale discussed above), as well as outright system crashes due to unsafe configuration recommendations.

In this paper, we propose a third avenue: cloud oracles. We observe that many decisions in the cloud are (yet another) scenario that can be viewed as a query optimization problem! This perspective on cloud decisions leads us to a little-used hammer in the query optimization toolbox: parametric query optimization (PQO).

Like machine learning, PQO splits optimization into an offline precomputation phase and an online serving phase. At compile time, PQO, based on a known cost model, materializes several execution plans, each one of which is optimal for a subset of all possible input parameters. At runtime, when the input parameters become known, PQO can then select the final, correct plan with near-zero overhead, as all relevant optimal plans have already been indexed [26]. PQO's success hinges on **three conditions**: 1) a relatively fixed cost model for long-lasting offline decisions; 2) a "well-behaved" cost model for tractable offline computation of all optimal decisions; 3) a tractable decision space—either enumerable in reasonable time, or collapsible/prunable into a tractable number of equivalence classes.

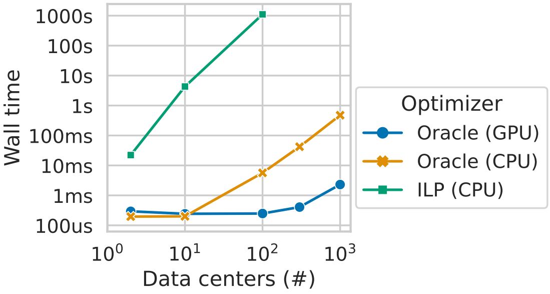

Figure 1: Optimization of object storage location for minimal access latency among *n* data centers—via the traditional exact approach (ILP) and our exact cloud oracle.

Cloud optimization broadly satisfies these conditions. Pricing and SLOs are reliable and fairly slow-changing by necessity, to maintain stable business relations and service operation. Also, the pay-per-use cloud billing model translates to simple linear cost functions: 0.3 per read and 0.5 per write for instance. Finally, the unbounded elasticity of cloud infrastructure prevents "bin-packing-style" pitfalls where due to limited capacity one partial decision changes the options available for the next partial decision—impeding pruning. As a result, the analogy from PQO to cloud oracles is quite direct: query plans map to system configurations, e.g., choices of VM or storage resources across regions or tiers; query execution costs map to price- and latency-based cost models; workload parameters like query selectivity map to API request metrics like PUT and GET rate.

Given the fixed cost model and the space of possible parameters, we propose to precompute a data structure that can be leveraged online to frequently compute the optimal cloud configuration for the current workload. To that end, it is natural in today's environment to consider an ML approach, and train a probabilistic or heuristic model for this task. However, given well-behaved cost functions and the financial penalties for misprediction in this setting, we propose instead to adopt an exact approach from computational geometry (§2.3): a convex polytope that represents all feasible and non-dominated configurations. This polytope is both compact (on the order of GBs) and deterministically optimal.

Calculating the optimal configuration for a given workload corresponds to finding the lowest point on the perimeter of the polytope matching each parameter. Conveniently, this can be computed quickly and with exact accuracy via a simple geometric calculation based on ray-shooting, described in §2.3. As the workload shifts, we can use the precomputed polytope and the fast online ray-shooting calculation to recompute the solution for the new input. Figure 1 highlights that the cloud oracle returns these optimization results within milliseconds rather than minutes—orders of magnitude speedup without loss of accuracy.

We refer to our approach as the *cloud oracle* approach because at runtime we are able to quickly extract exact answers from the data structure (oracle) we built in the offline step. Source code is available on GitHub: https://github.com/hydro-project/cloud_oracle.

2 FROM PARAMETRIC OPTIMIZATION TO CLOUD ORACLES

In the cloud, the design space is massive, we can reconfigure at will, and the stakes are high. Under these circumstances, there are two properties we want from a cloud optimization technique: *accuracy* and *velocity*. Accuracy refers to the distance the computed configuration is from the true optimal configuration. Velocity refers to the frequency with which we can recompute the optimal configuration online. In this section, we describe the two current approaches to optimization in the cloud: ILPs (high accuracy, low velocity) and ML (lower accuracy, high velocity). We argue for a third approach: explicit cloud oracles based on parametric optimization. This approach offers the best of both worlds for any problem amenable to a parametric representation.

2.1 The Rising Star: Machine Learning

The use of ML for decision-making in data systems has rapidly gone from a vision [1, 30] to a production reality [7, 42, 56, 57]. Fundamentally, machine learning does heavyweight offline computation (training) in order to achieve high-velocity online decision-making (inference). Its use cases in systems primarily replace the greedy or heuristic algorithms that were developed in prior decades. In many scenarios, we see big accuracy improvements from machine learning over previous heuristic-based approaches to approximation [50]. These solutions still fall short of perfect accuracy though and their improved accuracy over classical heuristics comes at a heavy price: computationally heavy training and lack of explainability.

First, real-world training of machine learning is highly burdensome. The process of gathering production telemetry at scale, testing and verifying these models, and preventing performance regressions has fostered the growth of an entirely new field of research and industry. Production *ML ops* is highly complex [2, 45] and with increasingly stringent privacy regulations, it is only getting more challenging [34].

Second, the lack of predictability and interpretability of decisions is a major barrier to production adoption of machine learning techniques. At Microsoft, researchers describe the production team's preference for heuristics over machine learning and for simple linear or tree models over the more complex models used in research settings [56, 57]. These preferences are due to the uninterpretability of models like deep learning models, which are often the models that outperform traditional solutions in research studies [50].

For problems that can be expressed analytically, parametric query optimization offers a workaround to the headaches of ML ops and gives exact and interpretable answers for all decisions. To understand the parametric approach, we must first understand the linear programming techniques that it is built on.

2.2 The Classicist: (Integer) Linear Programming

Linear programming (LP) is a mathematical optimization technique that has been in use since the mid-20th century. LP finds the input that minimizes some linear objective function, where the valid inputs are constrained by a set of linear equations. Many decision problems can be formulated in this way including data partitioning [9], view materialization [28], checkpoint selection [55], and resource allocation [40]. While continuous linear programming problems can be solved with a runtime complexity of approximately $O(n^3)$ [39], the integer versions of the problem are exponentially hard [29]. In Integer Linear Programming (ILP), only inputs with integer values are valid. Similarly, Mixed Integer Linear Programming (MILP) requires that some but not all of the inputs be integers. Both ILP and MILP problems take exponential time to solve—they are NP-hard [29].

Intuitively, ILP is harder than LP as the search space in ILP consists only of discrete points whereas LP considers smooth surfaces that can be walked, and whose intersections can be computed cheaply. In particular, the valid solutions to LP problems form a convex polytope. This polytope is the geometric shape enclosed by all the hyper planes (linear constraints) of the problem. Finding the intersection points on the perimeter of this polytope is sufficient for computing the optimal solution. Throughout this paper we will see that having this polytope representation of optimal solutions is very powerful, as we can use it not just to compute the answer on our current inputs efficiently, but to do what-if analysis and drift analysis efficiently as well (§3).

2.3 The New Kid: Explicit Cloud Oracles

In this work, we observe that parametric query optimization [10, 16, 26, 48, 49] and cloud optimization are extremely similar. As such, many of the ideas of PQO can be repurposed for efficient optimization in the cloud context. PQO not only has the potential to achieve both the accuracy of ILPs and the velocity of ML, but it also enables, as we describe next, highly expressive optimization tasks.

"Parametric query optimization attempts to identify at compile time several execution plans, each one of which is optimal for a subset of all possible values of the run-time parameters. At runtime, when the actual parameter values become known, it is then possible to choose the final optimal plan with essentially no overhead." [26] We propose transferring this idea to cloud optimization in the form of *cloud oracles*—a precomputed data structure of optimization decisions, akin to a multidimensional index of the parameter space, which enables lightning-fast online optimization queries.

Offline Optimization of a Cloud Oracle. As a first step in alleviating optimization overhead, we advocate for offline precomputation of cloud oracles based on PQO techniques. This is loosely analogous to the computation involved in training ML models, without any of the ML burdens of training data, model architecture selection, or hyperparameter tuning. PQO essentially comprises 1) *plan enumeration* in a fairly static problem setting and 2) a *linear cost model* allowing for efficient comparison of plans. Decisions in cloud optimization problems are equally enumerable and also have linear cost models. We can thus build a cloud oracle as the exhaustive collection of all optimal decisions over all possible parameters—enumerating and judging decisions similar to parametric query optimization.

Online Optimization by Geometry. Once the cloud oracle has been precomputed offline, the next step is querying the oracle in a way that is both fast and accurate. We can frame optimization tasks on a cloud oracle as computational geometry queries on a convex polytope. Rather than indexing the decisions, we point out that the exhaustive set of their linear cost functions forms a single convex polytope. We find that we can efficiently query the convex polytope even when the cloud oracle comprises millions of decisions with high-dimensional cost functions. This is in part thanks to the ability to reuse GPU hardware and linear algebra packages commoditized by ML. These can be used to perform highly optimized computational geometry queries on this convex polytope. Moreover, having access to the exhaustive set of all optimal decisions allows us to venture out to further optimization tasks beyond plain minimization tasks.

More formally, we represent the convex polytope as a dense matrix of hyperplanes derived from the cost functions of the decisions contained in the cloud oracle. That is, the linear cost function of a decision has the type $f : \mathbb{R}^n \to \mathbb{R}$, which can also be viewed as a hyperplane in $\mathbb{R}^{n+1}$ space—the $parameter \times cost$ space of the parameters and the resulting costs. The space enclosed by all these hyperplanes is the convex polytope and the surface of this convex polytope represents the lowest costs for all parameters.

In §4 we discuss the set of problems currently amenable to such cloud oracle constructions and the open research avenues to expanding that set of problems. But first, let us jump into a concrete example.

3 CASE STUDY: FAULT-TOLERANT OBJECT PLACEMENT

To make things concrete, we explore the situation of building a cloud oracle for fault-tolerant object storage and walk through three online decision scenarios: **which** configuration is optimal now, **what if** we were to make a large change to the workload, and **when** will we need to reconfigure in the future.

For our case study, consider an online retailer who uses object storage to manage its warehouse inventory—storing for each inventory item in each warehouse one object in Cloudflare's R2 object store [25]. The retailer wants to keep operating even in the face of disaster, e.g., power outage, but seeks to minimize access latency for their globally distributed clients. Specifically, the retailer wants to store two up-to-date copies of each object in two of Cloudflare's 300 data centers [14] that are at least 200km apart. Clients are permitted to read any one of the copies but must update both, so that the read latency is the minimum of the network latency between a client and the two data centers while the write latency is the maximum.

We define the fault-tolerant object placement problem for an object o as follows. Let $c \in C$ be the client data centers from which object o is accessed. Let $d \in D$ be the data centers that may host object o. Let $\vec{l}$ be the matrix with the network latency between clients and data centers. Let $\vec{w}$ and $\vec{r}$ be the workload vectors containing the write frequencies and read frequencies of all clients. If object o is hosted in data centers d and d', the overall access latency is calculated as in Equation 2. The objective is to minimize overall access latency while choosing two data centers with ≥200km distance:

Algorithm 1 ILP formulation for the optimal decisions of the fault-tolerant object placement problem with distance constraint. The distance constraint is linearized via auxiliary variables and constraints, as its straightforward formulation would be quadratic.

$$min \sum_{c \in C} z_c + \sum_{d \in D} l_{c,d} r_c y_{c,d} \text{ s.t.} \qquad \text{\#minimize total latency} \tag{5}$$

$$\sum_{d \in D} x_d = 2, \forall c \in C \qquad \text{\#write 2 DCs} \tag{6}$$

$$\sum_{d \in D} y_{c,d} = 1, \forall c \in C \qquad \text{\#read 1 DC} \tag{7}$$

$$x_d \geq y_{c,d}, \forall c \in C, \forall d \in D \qquad \text{\#read only if writing} \tag{8}$$

$$x_d, y_{c,d} \in \{0,1\}, \forall c \in C, \forall d \in D \qquad \text{\# no/yes decisions} \tag{9}$$

$$z_c \geq l_{c,d} r_c x_d, \forall d \in D, \forall c \in C \qquad \text{\#aux. write lat. } max(l_{c,d} r_c x_d) \tag{10}$$

$$z_c \in \mathbb{R}, \forall c \in C \qquad \text{\#aux. var for write latency} \tag{11}$$

$$200 v_{d,d'} \leq dist_{d,d'}, \forall d,d' \in D \qquad \text{\#linear dist. constraint} \tag{12}$$

$$2 v_{d,d'} \leq x_d + x_{d'}, \forall d,d' \in D \qquad \text{\#aux. mapping of } v \text{ to } x\text{:} \tag{13}$$

$$v_{d,d'} \geq x_d + x_{d'} - 1, \forall d,d' \in D \qquad \text{\#}v_{d,d'} \iff x_d \land x_{d'} \tag{14}$$

$$v_{d,d'} \in \{0,1\}, \forall d,d' \in D \qquad \text{\#aux. for dist. constraint} \tag{15}$$

$$\forall d,d' \in D : f_{d,d'} := \mathbb{R}^{|C|} \times \mathbb{R}^{|C|} \to \mathbb{R} \tag{1}$$

$$f_{d,d'}(\vec{w},\vec{r}) = \sum_{c \in C} max(\vec{l}_{c,d}, \vec{l}_{c,d'})\vec{w}_c + min(\vec{l}_{c,d}, \vec{l}_{c,d'})\vec{r}_c \tag{2}$$

$$\underset{d,d' \in D}{\text{argmin}} \quad f_{d,d'}(\vec{w},\vec{r}) \tag{3}$$

$$s.t. \quad dist(d,d') \geq 200 \tag{4}$$

The Classic ILP Formulation: Traditionally, one would encode this problem into the ILP in Algorithm 1. If you're not familiar with ILP formulations it is okay to ignore this diagram.

Overall, 0-1 variables x_d encode the binary decision of whether to store an object in a specific data center d and $y_{c,d}$ encodes the decision of which client reads which copy. The cost of these decisions in terms of write/read latency for the given write/read frequencies ($\vec{w}$, $\vec{r}$) and network latencies $\vec{l}$ is encoded in the linear coefficients $l_{c,d}$. However, the max terms and distance constraints do not permit straightforward linear encode. Their linear formulation requires auxiliary variables and constraints—Eq. 10–11 and Eq. 12–15, respectively. As a result, the ILP grows large and complex even for relatively few data centers, and we will see its prohibitive overhead in Section 3.2.

3.1 Offline Computation of the Cloud Oracle

Fortunately, we can compute a cloud oracle for the fault-tolerant object placement problem parameterized on the read and write frequencies (the workload which changes over time). The cloud oracle computation is possible because our three conditions all hold:

- *Condition 1–Fixed cost model*: Cloud vendors like Cloudflare offer reliable network latency [17, 18, 36]. In practice, we hence can consider the network latencies $\vec{l}$ as static coefficients rather than parameters in our cost function (Eq. 2).

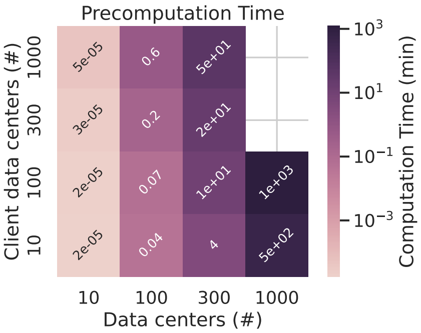

Figure 2: Worst-case single-threaded computation time for cloud oracles when scaling the number of data centers considered for storing objects and the number of client data centers determining the read/write frequency parameter dimensions.

- *Condition 2–Tractable cost model*: Our offline computation can easily resolve the *max* and *min* terms to plain linear coefficients (explained in Algorithm 2). This makes the cost model linear and tractable.
- *Condition 3–Tractable decision space*: Object stores promise elastic capacity for reasonable workloads [8, 13, 25, 44]. We can assume that objects do not compete for capacity, so that the decisions space collapses to the placement of a single object. Also, we can assume that the optimal data centers always have capacity, so that we can prune out non-optimal decisions without regret.

Algorithm 2 implements the offline computation of a cloud oracle for given client/storage data centers (C, D) and network latencies between those ($\vec{l}$). At a high level, we enumerate the data center pairs that satisfy the fault tolerance requirements and compute their read and write latencies. Then we prune out strictly non-optimal pairs and add the remaining pairs to a matrix representation. More specifically:

- *Lines 4–10*: We enumerate the entire search space, apply the constraint (Eq. 4), and importantly compute the *min* and *max* terms of the objective function to get the concise network latency $\vec{l}_p$ of each valid placement p.
- *Lines 12–15*: We now filter out irrelevant placement decisions. Without knowing the actual write and read frequencies ($\vec{w}$, $\vec{r}$), we can clearly tell that a placement decision never yields lowest total access latency when its latency coefficients are dominated ($\vec{l}_p > \vec{l}_{p'}$). The result is a compact yet accurate set of placement decisions that are optimal for some write/read frequencies.
- *Lines 17–19*: We finally construct the cloud oracle as a matrix containing the latencies of all optimal placement decisions. We reformulate the linear coefficients $\vec{l}_p$ as hyperplanes in $parameter \times latency$ space, adding an additional -1 coefficient. This is the textbook reformulation: $\vec{l}_p \cdot \vec{a} \Leftrightarrow x = \vec{l}_p \cdot \vec{a} \Leftrightarrow 0 = \vec{l}_p \cdot \vec{a} - x$, where $\vec{a} = [\vec{w}_0, \ldots, \vec{w}_{|C|}, \vec{r}_0, \ldots, \vec{r}_{|C|}]$. We now have a dense linear representation of all optimal placement decisions, allowing fast and accurate online optimization.

Algorithm 2 Algorithm for computing a cloud oracle for fault-tolerant object placement problem, given data centers D that may store objects, client data centers C that access objects, and the network latency $\vec{l}$ between those data centers.

```
1:  procedure COMPUTE_ORACLE($C, D, \vec{l}$)
2:      $placements \leftarrow \{\}$                          ▷ Initialize decisions of oracle
3:      ▷ Enumerate valid data center pairs as potential placements ◁
4:      for all $d, d' \in D$ do
5:          if $dist(d, d') \geq 200$ then                ▷ Check distance constraint
6:              $p \leftarrow (d, d')$                               ▷ Valid placement p
7:              ▷ Write/read latencies of p by element-wise max/min ◁
8:              $\vec{l}_p \leftarrow [max(l_{0,d}, l_{0,d'}), \ldots, max(l_{|C|,d}, l_{|C|,d'}),$
9:                    $min(l_{0,d}, l_{0,d'}), \ldots, min(l_{|C|,d}, l_{|C|,d'})]$
10:             $placements \leftarrow placements \bigcup (p, \vec{l}_p)$
11:     ▷ Filter out placements with strictly higher latency ◁
12:     for all $(p, \vec{l}_p), (p', \vec{l}_{p'}) \in placements$ do
13:         if $\vec{l}_{p'} < \vec{l}_p$ then
14:             $placements \leftarrow placements \setminus \{(p, \vec{l}_p)\}$
15:             break
16:     ▷ Convert into matrix of planes with coeff. $\vec{l}_p$ and extra -1 ◁
17:     $planes \leftarrow \begin{bmatrix} \vec{l}_{0,p} & \cdots & \vec{l}_{2|C|,p} & -1 \\ \vdots & & & \end{bmatrix}, \forall (p, \vec{l}_p) \in placements$
18:
19:     return Oracle($placements$, $planes$)
```

Figure 2 shows the worst-case computation time of Algorithm 2 under a range of problem sizes. Here, we scale the number of data centers that may store objects and the number of read/write frequency parameters of client data centers—scaling the decision space and the cost model dimensions. We report the computation time for single-threaded computation for the case that all possible data center pairs turn out to be optimal placement decisions. We can observe steeply increasing, as can be expected from the nested loops of the algorithm. Though, the single-threaded computation only takes 20 min at Cloudflare's scale (300 data centers). Beyond this scale, computation takes >7h where filtering of non-optimal placements dominates.

Implications: While seeming brute-force, the offline precomputation of the cloud oracle for fault-tolerant object placement is tractable. Enumerating the quadratic search space is manageable for the number of data centers that cloud vendors have today, and data-parallel computation will further offset overhead. We point out PQO techniques and further research avenues for precomputation of cloud oracles for more challenging cases in §4.

3.2 Efficient & Accurate Minimization

Which? Consider the processes of deciding which two data centers are the optimal places to store the copies of an object, as a function of recorded write/read frequencies ($\vec{w}$,$\vec{r}$) and the network latency $\vec{l}$. This maps directly to the typical minimization task of Eqs.3–4. Rather than solving the complex ILP of Algorithm 1, the existence of a cloud oracle allows solving this task orders of magnitude faster but equally accurately through simple geometric intersection search—namely *vertical ray-shooting*.

We illustrate vertical ray-shooting on a simplified oracle in Figure 3. We consider the "bounding box" (i.e., convex polytope) formed by the $parameter \times latency$ hyperplanes of all placement decisions

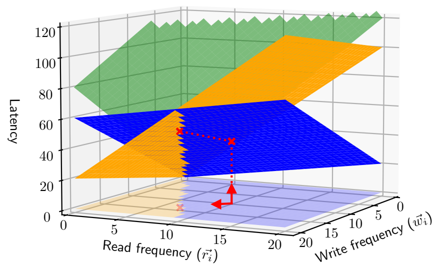

Figure 3: Illustration of vertical and directed ray-shooting onto the hyperplanes that represent the latency of decisions under any read/write frequency.

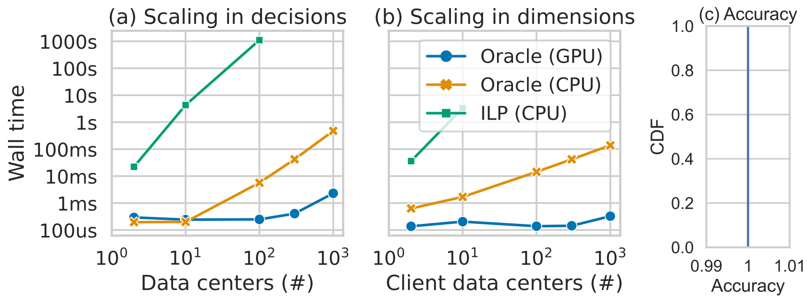

Figure 4: Minimization with the cloud oracle versus the ILP. (a) shows the scalability in the number of decisions in the search space (data centers D) under $|C|=300$ and (b) in the dimensions of the cost model (client data centers C) under $|D|=300$. (c) shows the accuracy of the cloud oracle relative to the ILP.

Algorithm 3 Algorithm for querying the oracle o for the optimal decisions with minimal latency under the given parameter $\vec{a}=[\vec{w},\vec{r}]$.

```
procedure QUERY_MINIMUM($o$, $\vec{a}$)
    ▷ Vertical ray-shooting: argmin on matrix-vector mul.    ◁
    $idx, value \leftarrow \text{argmin}(\overrightarrow{o.planes \cdot \vec{a}})$
    return $(o.placements_{idx}, value)$
```

in the oracle. Each point on the surface of this polytope is an optimum latency for some vector of input parameters $\vec{w}$ and $\vec{r}$. Given a specific input, we can start at the "floor" of the $parameter \times latency$ space where latency=0 (i.e., $[\vec{w},\vec{r},0]$), and compute the closest intersection point along a vertical line with the planes above. This point in $parameter \times latency$ space identifies the plane with the lowest latency for the given parameter—the optimal placement decision.

Algorithm 3 implements vertical ray-shooting with a single matrix-vector multiplication. As shown, vertical ray-shooting is a special case that allows aggressive simplification compared to the general ray-shooting used later. As such, the minimization task on the cloud oracle is very cheap and well suited for GPU-acceleration. Also, the results are exact by construction—given that the optimal decisions are contained in the cloud oracle.

Figure 4 shows the significant optimization speedup of minimization tasks with the cloud oracle compared to the traditional ILP. In these benchmarks we compare the optimization time of solving the ILP with Gurobi on a 40-core CPU and the cloud oracle on the CPU as well as a Nvidia V100 GPU[1].

Figure 4(a) shows the optimization time when scaling the number of data centers D and hence the number of feasible placement decisions under $|C|=300$—note the log scales. This affects the complexity of solving the ILP and the size of the cloud oracle. We see the high optimization overhead of the ILP, starting at >10ms for only 2 data centers and reaching >1000s for 100 data centers. Even for 1000 data centers, the cloud oracle remains sub second on the CPU and sub 10 milliseconds on the GPU. The cloud oracle is more than 4 orders of magnitude faster than the ILP.

Figure 4(b) shows the optimization time when instead scaling the number of client data centers $|C|$, i.e., the number of parameters of the write/read frequencies in the model, under $|D| = 300$. Also under high-dimensional parameters, the cloud oracle is orders of magnitude faster.

Finally, Figure 4(c) shows the accuracy of the cloud oracle compared to the ILP for both of the above experiments. The cloud oracle is perfectly accurate—it yields exactly the same optimization results.

Implications: Cloud oracles are fast while achieving exact accuracy! Minimization tasks that use the cloud oracle are orders of magnitude faster than ILPs, even if the cloud oracle has to cover a large search space, e.g., all 300 Cloudflare data centers, and the cost functions have hundreds of parameter dimensions. Using cloud oracles makes accurate minimization affordable at cloud scale and benefits from accelerator hardware.

3.3 Efficient Scenario Planning

What If? Consider now a situation in which our online retailer wants to expand from the US to the EU and has to negotiate discounted contracts for reserving capacity in specific EU data centers. The retailer must explore "what if" scenarios to determine how the choice of data centers will change the access patterns and thus affect optimal object placements and access latencies. Such scenario planning is based on "what if" analyses, which typically uses stochastic simulation to iteratively explore a vast number of possible scenarios to present anticipated latency profiles.

Simulation techniques like Monte Carlo simulation essentially consist of an outer loop that draws sample parameters from a trace or model, and an inner loop that computes the optimal decision and resulting costs for a given sample [31]. These simulations need to evaluate many samples to confidently judge scenarios. This is infeasible if the inner loop is an ILP that takes seconds or even minutes to compute. Our cloud oracle approach can be used as an alternative lightning-fast inner loop, allowing these simulations to do what they are intended to do: aggressively explore many samples and scenarios.

Algorithm 4 shows that Monte Carlo simulation immediately follows the implementation of the minimization task. Notably, besides solving for a single sample quickly through vectorization, this simulation also easily scales across several GPUs, simply by replicating the cloud oracle matrices and partitioning the samples.

[1] ILP solvers rely on sequential refinement of solutions that preclude parallelism. Commercial solvers thus operate only on CPUs and do not support GPU acceleration [19].

Algorithm 4 Algorithm for simple Monte Carlo simulation computing the expected latency improvement between two scenarios. o is the cloud oracle for the US-only scenario and o' the cloud oracle for the expansion to US+EU.

```
procedure SIMULATE_SCENARIO($o, o', Samples$)
    $improvements \leftarrow \{\}$   ▷ Latency improvements of $\vec{a} \in Samples$
    for all $\vec{a} \in Samples$ do
        ▷ Ratio of new optimal latency vs. prior optimal latency ◁
        $i \leftarrow min(o'.planes \cdot \vec{a}) / min(o.planes \cdot \vec{a})$
        $improvements \leftarrow improvements \bigcup \{i\}$
    return $mean(improvements), confidence(improvements)$
```

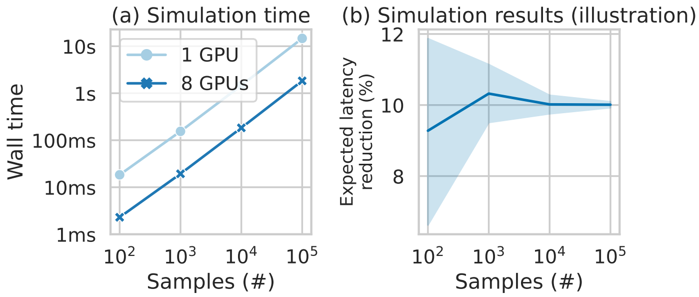

Figure 5: Monte Carlo simulation with the cloud oracle for 300 data centers (D) and client data centers (C). (a) Scalability of the cloud oracle in the number of samples optimized on 1 GPU and 8 GPUs. (b) An illustration how simulation results (median and 95-confidence interval) converge under the number of samples from a trace.

Figure 5 demonstrates the Monte Carlo simulation on top of the cloud oracle for judging the latency improvement when expanding from the US to the EU. We randomly generate access frequency samples and compute the expected latency reduction at the scale of Cloudflare ($|D| = |C| = 300$). Figure 5(a) shows that the simulation time scales linearly in the number of samples and GPUs. Figure 5(b) illustrates that thousands of samples from a trace may be necessary for the median and 95-confidence-interval to converge. In combination, we can see that the cloud oracle on the GPU can provide confident simulation results within milliseconds to seconds—interactive response time.

Implications: The cloud oracle approach makes stochastic simulation feasible at the scale of the modern clouds. This unlocks a wide range of planning and forecasting scenarios. One interesting family of applications are system configuration tasks like storage placement or other system parameter ("knob") tuning. A wide range of performance profiles can be created with very low overhead and either presented visually to a human for decision-making, or fed into an automatic configuration service. In either case, decisions can be made to account for a wide variety (or a probability distribution) of "what if" scenarios, each one accurately assessed via the cloud oracle. Stochastic methods are not only interesting for applications on top of cloud oracles, but are also interesting avenues for the construction of robust cloud oracles over uncertain cost models, as we point out in §4.

Algorithm 5 Algorithm for querying the oracle o for the next optimal decision under the drift $\vec{d}$ of parameter $\vec{a}$.

```
procedure QUERY_DRIFT_DIRECTED($o, \vec{a}, \vec{d}$)
    $curr, cost \leftarrow$ QUERY_MINIMIUM($o, \vec{a}$)
    $\overrightarrow{plane} \leftarrow \overrightarrow{o.planes}_{curr}$   ▷ Plane of current optimum
    $\vec{s} \leftarrow [\vec{a}, cost]$   ▷ Start point on plane
    $\vec{d}' \leftarrow \vec{d} - ((\vec{d} \cdot \overrightarrow{plane}) * \overrightarrow{plane})$   ▷ Projected drift along plane
    $\overrightarrow{Other} \leftarrow \overrightarrow{o.planes} \setminus \{\overrightarrow{plane}\}$   ▷ Matrix of remaining planes
    ▷ Ray-shooting from start $\vec{s}$ in direction of projected drift $\vec{d}'$ ◁
    $\overrightarrow{distance} \leftarrow (\overrightarrow{Other} \cdot -\vec{s}) / (\overrightarrow{Other} \cdot \vec{d}')$
    for all $i \in 0..n$ do   ▷ Find closest positive distance
        if $\overrightarrow{distance}_i > 0$ && $\overrightarrow{distance}_i < min\_dist$ then
            $min\_idx \leftarrow i, min\_dist \leftarrow \overrightarrow{distance}_i$
    $\overrightarrow{closest} \leftarrow \vec{s} + (\vec{d}' * min\_dist)$   ▷ Closest intersection point
    $next \leftarrow o.decisions_{min\_idx}$   ▷ Next optimal decision
    $\overrightarrow{param\_next} \leftarrow \overrightarrow{closest}_{0,...,n-1}$   ▷ Next parameter vector
    $cost\_next \leftarrow \overrightarrow{closest}_n$   ▷ Next latency
    return $(next, \overrightarrow{param\_next}, cost\_next, min\_dist)$
```

3.4 Efficient Migration Planning

When? Finally, consider the task of migration planning for large objects with changing access pattern, e.g., due to diurnal or seasonal popularity in different regions of the globe [22, 47]. Our retailer may want to migrate objects to improve access latency. However, migrating large objects takes time. Suppose the online retailer monitors the access pattern and can estimate the migration duration (x), knowing the available network bandwidth. They want to start migrating at $t - x$, where t is the *time when the current placement starts to become suboptimal.* But when is time t?

With current approaches, one would have to solve a series of ILPs over quantized time steps to find time t. This exacerbates the already high overhead, forcing current large-scale industrial systems to take a reactive migration approach [6]. In this situation, we can take advantage of the geometric nature of the cloud oracle to answer new classes of queries.

Planning under predictable drift. Consider predictable diurnal access pattern. Our retailer may want to migrate objects to evening locations, when people get home and have time for online shopping. In this situation, the cloud oracle can answer the query: *How far can the parameters drift in a given direction without changing the optimum?* If we know the direction and rate of change, we can use such a query to tell us precisely when the current optimum will no longer apply; this allows us to plan ahead for that eventuality.

We can directly compute when to migrate under given drift via *directed ray-shooting* on the hyperplanes of the convex polytope representing the cloud oracle. As illustrated in Figure 3, we can view the problem as a point in *parameter*×*latency* space that moves in the direction and speed of a given parameter drift vector. The point moves along the hyperplane of the current optimum and eventually collides with a neighboring hyperplane—at *parameter*×*latency* coordinates where the current optimum has the same latency as its neighbor. This intersection point identifies the next optimal decision including the parameters, latency (cost), and time when this next optimum is reached. Due to linearity and convexity, linear intersection search in the direction of the drift suffices to find the intersection point.

Algorithm 5 implements the query for directed drift. It first projects the given drift $\vec{d}$ onto the current optimal hyperplane in $parameter \times latency$ space and then computes the closest intersection with the remaining hyperplanes from the current $parameter \times latency$ point.

Planning under undirected drift.

Now consider predictable access peaks. Before events like Black Friday, our retailer must plan migration of all their objects in advance. Based on historic information, the retailer can estimate the access frequency of each object and anticipate its optimal placement under some confidence. One may apply Monte Carlo simulation but in this situation the simulation for each object would require significant computation time, even with the cloud oracle. The cloud oracle offers a shortcut via the query: *What is the least parameter drift that changes the optimal configuration?* This query tells us if migration of an object will be necessary. If the confidence interval is smaller than the output drift, then the object will have a single optimal placement. Otherwise, computationally intensive simulation needs to anticipate a migration plan.

We can efficiently compute closest neighboring optimum in any drift direction via intersection search—after relaxing the problem. This query can be viewed as a point drifting within the current hyperplane and we have to find the closest intersection point with the remaining hyperplanes in *any drift direction*. We are not given a drift vector, instead we have to compute the vector $\vec{d}^*$ that lies inside the current hyperplane and minimizes the distance from the current point on x_0 to some intersection point x^*:

$$min \quad t\text{, s.t.} \qquad \text{\# minimal distance } t \tag{16}$$

$$x_0 + t\vec{d}^* = x^* \wedge \exists p \in P : x^* \in p \qquad \text{\# intersection with } p_i \tag{17}$$

$$\vec{d}^* \cdot \vec{n} = 0 \qquad \text{\# parallel to current plane} \tag{18}$$

$$||\vec{d}^*||_2 = 1 \qquad \text{\# normalized distance} \tag{19}$$

Rather than falling back to an expensive solver, we can utilize the cloud oracle when relaxing the problem. That is, we can utilize *Lagrangian relaxation* to formulate an unconstrained minimization problem for the closest intersection to each neighboring hyperplane individually. We can efficiently compute this relaxed problem and then simply take the minimum to find the overall closest intersection—the least drift that changes the current optimum. Appendix A details our relaxation that Algorithm 6 implements to answer the undirected drift query.

Figure 6 shows the optimization time of the advanced drift queries in comparison to the simple minimization task. Here, we repeat the scaling experiment from earlier. We can see that even for these advanced optimization tasks the optimization time remains sub 10 milliseconds on the GPU and overall very manageable compared to the ILP solving the simple minimization.

Algorithm 6 Algorithm for querying the oracle for the next optimal decision under unknown drift of parameter $\vec{a}$.

```
procedure QUERY_DRIFT_UNDIRECTED($o$: Oracle, $\vec{a}$: Parameter)
    $curr, cost \leftarrow$ QUERY_MINIMIUM$(o, \vec{a})$
    $\overrightarrow{plane} \leftarrow o.\overrightarrow{planes}_{curr}$          ▹ Plane of current optimum
    $\overrightarrow{Other} \leftarrow o.\overrightarrow{planes} \setminus \{\overrightarrow{plane}\}$     ▹ Matrix of remaining planes
    $\vec{s} \leftarrow [\vec{a}, cost]$   ▹ Start point at parameter and cost coordinates
    ▹ Solving closest intersection for each plane via relaxation ◃
    for all $\vec{p}_i \in \overrightarrow{Other}$ do
        $\alpha_i \leftarrow \vec{p}_i \cdot \overrightarrow{plane}$
        $\vec{v}_i \leftarrow (\alpha_i \overrightarrow{plane} - \vec{p}_i) / \sqrt{1 - \alpha_i^2}$
        $distance_i \leftarrow (\vec{p}_i \cdot -\vec{s}) / (\vec{p}_i \cdot \vec{v}_i)$
    $min\_idx, min\_dist \leftarrow \text{argmin}(\overrightarrow{distance})$
    $\overrightarrow{closest} \leftarrow \vec{s} + (\vec{V}_{min\_idx} * min\_dist)$      ▹ Closest intersection
    $next \leftarrow o.decisions_{min\_idx}$                 ▹ Next optimal decision
    $\overrightarrow{param\_next} \leftarrow \overrightarrow{closest}_{0,\dots,n-1}$          ▹ Next parameter vector
    $cost\_next \leftarrow \overrightarrow{closest}_n$                          ▹ Next latency
    return $(next, \overrightarrow{param\_next}, cost\_next, min\_dist)$
```

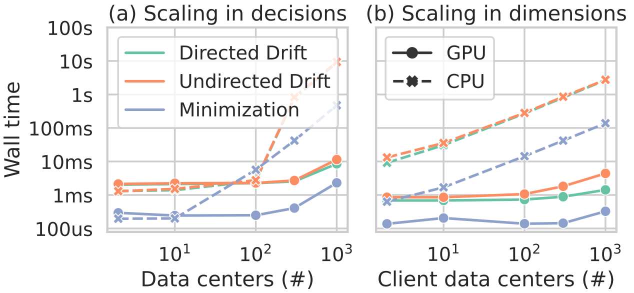

Figure 6: Scalability of the drift optimization tasks compared to the simple minimization on the cloud oracle. (a) shows the scalability in the number of decisions in the search space (data centers D) under $|C| = 300$ and (b) in the dimensions of the cost model (client data centers C) under $|D| = 300$.

Implications: The low-overhead drift queries highlight the power of geometric computations on the cloud oracle and indicate interesting applications in dynamic optimization. Not only do drift queries allow us to plan ahead for a given drift, but they may also benefit time series-based dynamic optimization. A common approach is to predict a time series of workload parameters and then optimize over quantized time slots [12, 20, 37, 43, 51]. Fine-grained time slots allow for high accuracy but suffer from high overhead while coarse-grained time slots minimize overhead but also worsen accuracy. Instead, traveling the surface of the time series and the cost functions has the potential to offer high accuracy and avoid high overhead (see annealing methods described in §4.2).

4 TOWARD CLOUD ORACLES

We have one major question left to address: For what kind of problems can we compute a cloud oracle? Akin to PQO, offline optimization for a cloud oracle requires enumeration of candidate decisions and subsequent selection of optimal decisions based on a cost model [16, 49]. The three conditions enabling efficient offline optimization are: 1) a relatively fixed cost model; 2) a mostly linear cost model; 3) independence of decisions.

The placement problem for Durable Objects fulfills these three conditions and has a small search space. This enables straightforward enumeration and selection of decisions for the cloud oracle. We have seen that the resulting cloud oracle offers large speedups and improved accuracy for a range of important optimization tasks. Indeed, this beneficial scenario applies to placement problems that do not require replication and have elastic capacity. For example, cost optimization of S3 objects has a linear pay-per-use pricing model and a search space of 99 AZs with each 6 storage tiers (594 decisions) [4, 5].

In contrast, "bin-packing-style" problems violate condition 3, hence making offline optimization extremely difficult. For example, consider data placement to minimize average access latency for a DBMS that runs on a VM with local and remote storage. Local storage has low access latency but limited capacity, so we must decide which tables to pack in there. For optimal placement, we must jointly decide for all tables at once, which blows up the problem with little chance of pruning. Compared to independent decisions, this bin-packing problem appears combinatorially intractable to precompute offline.

Along the following three research questions, we now discuss problems that challenge the application of cloud oracles but offer promising directions:

(1) How to make enumeration of candidate decisions tractable in large configuration spaces of cloud-scale systems?
(2) How to select optimal decisions under non-linear cost models?
(3) How to select optimal decisions under variance in cost models, e.g., network latency?

4.1 Enumeration in Large Search Spaces

Placement *with replication* still fits our three conditions, but optimizing for combinations of placements blows up the search space. For example, consider geo-replication of objects for latency minimization or geo-replication of DBMSs for fault-tolerance. These problems challenge efficient candidate enumeration. Facing a massive search space, one important lesson to learn from (parametric) query optimization is that effective pruning is imperative [10, 33, 48]. No matter how efficient the selection, excessive enumeration of candidates will prohibit tractable offline optimization. Pruning is critical, but it is heuristic. The challenge is that pruning must exploit problem structure, making algorithms hard to design and their performance case-sensitive.

An interesting direction for effective pruning in the cloud is algorithmic meta-optimization [15, 21, 53]. Given the scale and diversity of the cloud, manual design and selection of pruning algorithms is challenging. Instead, meta-optimization of pruning algorithms could be an approach for automatic classification of problem structure and algorithm specialization. Algorithmic meta-optimization is an ongoing research effort and may benefit from cost estimation and other techniques à la query optimization.

4.2 Decisions Selection under Non-linear Costs

Life is not always linear, and neither are cost models. For example, state-of-the-art cost models for runtime estimation are learned models. For application to these problems the computation of cloud oracles has to support learned cost models. These cost models precisely capture complex relationships between workload parameters and costs, as shown for cardinality and cost estimation [23, 24, 38, 50]. However, this complicates the navigation of their cost surface, required for selecting the optimal decisions for the cloud oracle.

Gradient descent, annealing, and differential evolution [26, 41, 46] offer directions for computing cloud oracles on models with complex cost surfaces. Convex linear models allow efficient computation in cloud oracles, as their smooth surfaces obviate the optimal lowest-cost decisions. Learned models have rough surfaces with hills that obfuscate lowest-cost decisions. A simple idea to compute optimal decisions under complex cost models is the pair-wise comparison of candidates, similar to the multi-objective PQO approach [48]. That is, the cost of two candidates are the learned functions $f^*(\vec{a})$ and $g^*(\vec{a})$. The cost difference between the two candidates is the function $h(\vec{a})=f^*(\vec{a})-g^*(\vec{a})$. If h has a negative value, there exists a parameter for which f^* is cheaper than g^* otherwise f^* is dominated and never optimal. We can use annealing methods—that Ioannidis et al. [26] originally proposed for PQO—to search the surface of h for negative values. Afterwards, we can construct a cloud oracle with a compact model as lookup structure for selected optimal decisions. Research on robust knowledge distillation of a compact "student model" for given decisions is required, see [3]. Also, efficient methods for computing optimal decisions are required for problems with large parameter and search spaces.

Note that annealing techniques are similarly interesting for solving complex optimization tasks on the surface of the cost model, like our drift query.

4.3 Decisions Selection under Uncertainty

Random variables that capture uncertainty challenge the static condition (1) for offline optimization. For example, placement of objects onto edge locations may involve significant variation in network latency. Also, monitoring of workload parameters like access frequency in a large distributed systems will not be exact, but may involve uncertainty due to sampling. Particular challenges are how to derive a probabilistic cost model with random cost coefficients and how to select optimal decisions offline in the presence of random cost coefficients and random workload parameters.

Interesting directions are learned cost models capturing uncertainty and robust parametric optimization. For example, one may explore the transfer of learned cardinality estimation with uncertainty [35]. A starting point for robust optimization is the work [32], which considers robust load distribution for distributed streaming systems. Offloading robust optimization to the precomputation of cloud oracles not only leads to fast and robust online optimization for cloud systems, but may also provide opportunities for offline verification. The behavior of online optimization under uncertainty as well as pure ML-based optimization is difficult to foresee. The materialized decisions in a cloud oracle instead are known explicitly, thus enabling more stringent offline verification.

CONCLUSION

The cloud is both powerful and complex. To harness its power to the fullest, we need to make accurate and rapid choices in enormous decision spaces. Classical ILP does this accurately, but slowly. Machine learning approaches trade accuracy for speed,while often introducing ongoing costs of ML ops. We have proposed a best-of-both-worlds design, cloud oracles. This approach follows the ML script of slow offline training combined with fast online inference but does so with the data-independent explainable solutions of ILP.

REFERENCES

[1] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, et al. 2020. The Seattle report on database research. *ACM Sigmod Record* 48, 4 (2020), 44–53.

[2] Ashvin Agrawal, Rony Chatterjee, Carlo Curino, Avrilia Floratou, Neha Gowdal, Matteo Interlandi, Alekh Jindal, Kostantinos Karanasos, Subru Krishnan, Brian Kroth, et al. 2019. Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. *arXiv preprint arXiv:1909.00084* (2019).

[3] Zeyuan Allen-Zhu and Yuanzhi Li. 2022. Towards Understanding Ensemble, Knowledge Distillation and Self-Distillation in Deep Learning. https://openreview.net/forum?id=Uuf2q9TfXGA

[4] Inc. or its affiliates Amazon Web Services. 2023. *Amazon S3 Storage Classes.* https://aws.amazon.com/s3/storage-classes/

[5] Inc. or its affiliates Amazon Web Services. 2023. *AWS Global Infrastructure.* https://aws.amazon.com/about-aws/global-infrastructure/

[6] Muthukaruppan Annamalai, Kaushik Ravichandran, Harish Srinivas, Igor Zinkovsky, Luning Pan, Tony Savor, David Nagle, and Michael Stumm. 2018. Sharding the Shards: Managing Datastore Locality at Scale with Akkio. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 445–460. https://www.usenix.org/conference/osdi18/presentation/annamalai

[7] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J Green, Monish Gupta, Sebastian Hillig, et al. 2022. Amazon Redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data*. 2205–2217.

[8] Microsoft Azure. 2023. *Azure Blob Storage.* https://azure.microsoft.com/en-us/products/storage/blobs/

[9] Nirvik Baruah, Peter Kraft, Fiodar Kazhamiaka, Peter Bailis, and Matei Zaharia. 2022. Parallelism-Optimizing Data Placement for Faster Data-Parallel Computations. *Proceedings of the VLDB Endowment* 16, 4 (2022), 760–771.

[10] Nicolas Bruno and Rimma V. Nehme. 2008. Configuration-parametric query optimization for physical design tuning. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*. Association for Computing Machinery, New York, NY, USA, 941–952. https://doi.org/10.1145/1376616.1376710 .

[11] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E. Gonzalez, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, Michael W. Mahoney, Aditya Parameswaran, David Patterson, Raluca Ada Popa, Koushik Sen, Scott Shenker, Dawn Song, and Ion Stoica. 2022. The Sky Above The Clouds. https://doi.org/10.48550/arXiv.2205.07147 .

[12] Niangjun Chen, Joshua Comden, Zhenhua Liu, Anshul Gandhi, and Adam Wierman. 2016. Using Predictions in Online Optimization: Looking Forward with an Eye on the Past. *ACM SIGMETRICS Performance Evaluation Review* 44, 1 (June 2016), 193–206. https://doi.org/10.1145/2964791.2901464

[13] Google Cloud. 2023. *Cloud Storage.* https://cloud.google.com/storage/

[14] Inc. Cloudflare. 2023. *The Cloudflare global network.* https://www.cloudflare.com/network/

[15] Joshua Comden, Sijie Yao, Niangjun Chen, Haipeng Xing, and Zhenhua Liu. 2019. Online Optimization in Cloud Resource Provisioning: Predictions, Regrets, and Algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 1 (March 2019), 16:1–16:30. https://doi.org/10.1145/3322205.3311087

[16] Lyric Doshi, Vincent Zhuang, Gaurav Jain, Ryan Marcus, Haoyu Huang, Deniz Altinbüken, Eugene Brevdo, and Campbell Fraser. 2023. Kepler: Robust Learning for Parametric Query Optimization. *Proceedings of the ACM on Management of Data* 1, 1 (May 2023), 109:1–109:25. https://doi.org/10.1145/3588963 .

[17] Mike Freemon. 2022. *Optimizing TCP for high WAN throughput while preserving low latency.* https://blog.cloudflare.com/optimizing-tcp-for-high-throughput-and-low-latency/

[18] Vasilis Giotsas and Marwan Fayed. 2021. *"Look, Ma, no probes!" — Characterizing CDNs' latencies with passive measurement.* https://blog.cloudflare.com/cdn-latency-passive-measurement/

[19] Greg Glockner. 2023. *Does Gurobi support GPUs?* https://support.gurobi.com/hc/en-us/articles/360012237852-Does-Gurobi-support-GPUs

[20] Gautam Goel, Niangjun Chen, and Adam Wierman. 2017. Thinking fast and slow: Optimization decomposition across timescales. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 1291–1298. https://doi.org/10.1109/CDC.2017.8263834

[21] Justin Gottschlich, Armando Solar-Lezama, Nesime Tatbul, Michael Carbin, Martin Rinard, Regina Barzilay, Saman Amarasinghe, Joshua B. Tenenbaum, and Tim Mattson. 2018. The three pillars of machine programming. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2018)*. Association for Computing Machinery, New York, NY, USA, 69–80. https://doi.org/10.1145/3211346.3211355

[22] Raúl Gracia-Tinedo, Yongchao Tian, Josep Sampé, Hamza Harkous, John Lenton, Pedro García-López, Marc Sánchez-Artigas, and Marko Vukolic. 2015. Dissecting UbuntuOne: Autopsy of a Global-Scale Personal Cloud Back-End. In *Proceedings of the 2015 Internet Measurement Conference (IMC '15)*. Association for Computing Machinery, New York, NY, USA, 155–168. https://doi.org/10.1145/2815675.2815677 .

[23] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proceedings of the VLDB Endowment* 15, 11 (July 2022), 2361–2374. https://doi.org/10.14778/3551793.3551799

[24] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: learn from data, not from queries! *Proceedings of the VLDB Endowment* 13, 7 (March 2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[25] Cloudflare Inc. 2023. *Cloudflare R2.* https://www.cloudflare.com/products/r2/

[26] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. 1997. Parametric query optimization. *The VLDB Journal* 6, 2 (May 1997), 132–151. https://doi.org/10.1007/s007780050037 .

[27] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. 2023. Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays. 1375–1389. https://www.usenix.org/conference/nsdi23/presentation/jain

[28] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, and Hiren Patel. 2018. Selecting subexpressions to materialize at datacenter scale. *Proceedings of the VLDB Endowment* 11, 7 (2018), 800–812.

[29] Richard M Karp, RE Miller, and JW Thatcher. 1972. Reducibility among combinatorial problems, Complexity of computer computations. *Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, NY, 1972* 378476, 51 (1972), 14644.

[30] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H Chi, Jialin Ding, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2021. Sagedb: A learned database system. (2021).

[31] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev. 2014. Why the Monte Carlo method is so important today. *WIREs Computational Statistics* 6, 6 (2014), 386–392. https://doi.org/10.1002/wics.1314 .

[32] Chuan Lei and Elke A. Rundensteiner. 2014. Robust Distributed Query Processing for Streaming Data. *ACM Transactions on Database Systems* 39, 2 (May 2014), 17:1–17:45. https://doi.org/10.1145/2602138

[33] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query optimization through the looking glass, and what we found running the Join Order Benchmark. *The VLDB Journal* 27, 5 (Oct. 2018), 643–668. https://doi.org/10.1007/s00778-017-0480-7

[34] Weixin Liang, Girmaw Abebe Tadesse, Daniel Ho, L Fei-Fei, Matei Zaharia, Ce Zhang, and James Zou. 2022. Advances, challenges and opportunities in creating data for trustworthy AI. *Nature Machine Intelligence* 4, 8 (2022), 669–677.

[35] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. 2021. Fauce: fast and accurate deep ensembles with uncertainty for cardinality estimation. *Proceedings of the VLDB Endowment* 14, 11 (July 2021), 1950–1963. https://doi.org/10.14778/3476249.3476254

[36] Marek Majkowski. 2015. *The story of one latency spike.* https://blog.cloudflare.com/the-story-of-one-latency-spike/

[37] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. 2019. Cost Optimization for Dynamic Replication and Migration of Data in Cloud Data Centers. *IEEE Transactions on Cloud Computing* 7, 3 (2019), 705–718. https://doi.org/10.1109/TCC.2017.2659728

[38] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1275–1288. https://doi.org/10.1145/3448016.3452838

[39] Nimrod Megiddo. 1987. On the complexity of linear programming. In *Advances in Economic Theory* (1 ed.), Truman Fassett Bewley (Ed.). Cambridge University Press, 225–268. https://doi.org/10.1017/CCOL0521340446.006

[40] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 521–537.

[41] Godfrey C. Onwubolu and B. V. Babu. 2004. *New Optimization Techniques in Engineering*. Studies in Fuzziness and Soft Computing, Vol. 141. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39930-8

[42] Conor Power, Hiren Patel, Alekh Jindal, Jyoti Leeka, Bob Jenkins, Michael Rys, Ed Triou, Dexin Zhu, Lucky Katahanas, Chakrapani Bhat Talapady, et al. 2021. The cosmos big data platform at Microsoft: over a decade of progress and a decade to look forward. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3148–3161.

[43] James Blake Rawlings, David Q. Mayne, and Moritz Diehl. 2017. *Model predictive control: theory, computation, and design* (2nd edition ed.). Nob Hill Publishing, Madison, Wisconsin.

[44] Amazon Web Services. 2023. *Cloud Object Storage.* https://aws.amazon.com/s3/

[45] Shreya Shankar, Rolando Garcia, Joseph M Hellerstein, and Aditya G Parameswaran. 2022. Operationalizing machine learning: An interview study. *arXiv preprint arXiv:2209.09125* (2022).

[46] Rainer Storn and Kenneth Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (Dec. 1997), 341–359. https://doi.org/10.1023/A:1008202821328

[47] Rebecca Taft, Nosayba El-Sayed, Marco Serafini, Yu Lu, Ashraf Aboulnaga, Michael Stonebraker, Ricardo Mayerhofer, and Francisco Jose Andrade. 2018. P-Store: An Elastic Database System with Predictive Provisioning. *Proceedings of the 2018 International Conference on Management of Data* (2018).

[48] Immanuel Trummer and Christoph Koch. 2017. Multi-objective parametric query optimization. *Commun. ACM* 60, 10 (Sept. 2017), 81–89. https://doi.org/10.1145/3068612 .

[49] Kapil Vaidya, Anshuman Dutt, Vivek Narasayya, and Surajit Chaudhuri. 2021. Leveraging query logs and machine learning for parametric query optimization. *Proceedings of the VLDB Endowment* 15, 3 (Nov. 2021), 401–413. https://doi.org/10.14778/3494124.3494126 .

[50] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are we ready for learned cardinality estimation? *Proceedings of the VLDB Endowment* 14, 9 (2021), 1640–1654.

[51] Tyler Westenbroek, Max Simchowitz, Michael I. Jordan, and S. Shankar Sastry. 2021. On the Stability of Nonlinear Receding Horizon Control: A Geometric Perspective. In *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, Austin, TX, USA, 742–749. https://doi.org/10.1109/CDC45484.2021.9682955

[52] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. 2023. SkyPilot: An Intercloud Broker for Sky Computing. 437–455. https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng

[53] Wenjie Yi, Rong Qu, and Licheng Jiao. 2023. Automated algorithm design using proximal policy optimisation with identified features. *Expert Systems with Applications* 216 (April 2023), 119461. https://doi.org/10.1016/j.eswa.2022.119461

[54] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards dynamic and safe configuration tuning for cloud databases. In *Proceedings of the 2022 International Conference on Management of Data*. 631–645.

[55] Yiwen Zhu, Matteo Interlandi, Abhishek Roy, Krishnadhan Das, Hiren Patel, Malay Bag, Hitesh Sharma, and Alekh Jindal. 2021. Phoebe: a learning-based checkpoint optimizer. *arXiv preprint arXiv:2110.02313* (2021).

[56] Yiwen Zhu, Subru Krishnan, Konstantinos Karanasos, Isha Tarte, Conor Power, Abhishek Modi, Manoj Kumar, Deli Zhang, Kartheek Muthyala, Nick Jurgens, et al. 2021. Kea: Tuning an exabyte-scale data infrastructure. In *Proceedings of the 2021 International Conference on Management of Data*. 2667–2680.

[57] Yiwen Zhu, Yuanyuan Tian, Joyce Cahoon, Subru Krishnan, Ankita Agarwal, Rana Alotaibi, Jesús Camacho-Rodriguez, Bibin Chundatt, Andrew Chung, Niharika Dutta, et al. 2023. Towards Building Autonomous Data Services on Azure. In *Companion of the 2023 International Conference on Management of Data*. 217–224.

Appendix A UNDIRECTED DRIFT: LAGRANGIAN RELAXATION

We present a derivation to compute the closest intersection point from a given parameter along the current optimal plane, as utilized in Section 3.4. Figure 7 illustrates two hyperplanes in $\mathbb{R}^n$, denoted as P_0 and P_i, each defined by their respective unit normal vectors $\boldsymbol{n}_0$ and $\boldsymbol{n}_i$. Consider a point $\boldsymbol{x}_0$ on P_0 and a unit vector $\boldsymbol{v}_0$ originating from $\boldsymbol{x}_0$ and lying tangentially to P_0. Importantly, $\boldsymbol{v}_0$ is perpendicular to $\boldsymbol{n}_0$. Our goal is to determine the intersection point $\boldsymbol{x}_i$ on P_i, where a ray in the direction of $\boldsymbol{v}_0$ intersects P_i.

The intersection point $\boldsymbol{x}_i$ can be expressed as

$$\boldsymbol{x}_i = \boldsymbol{x}_0 + t\boldsymbol{v}_0, \tag{A.1}$$

where t is a scalar. To find t, we consider an arbitrary point $\boldsymbol{o}$ on P_i and the orthogonality of $\boldsymbol{n}_i$ with P_i, giving $(\boldsymbol{x}_i - \boldsymbol{o}) \cdot \boldsymbol{n}_i = 0$. Assuming P_i passes through the origin, we can simplify the formulation by setting $\boldsymbol{o}$ to the origin, leading to $\boldsymbol{x}_i \cdot \boldsymbol{n}_i = 0$. Taking the dot product of equation (A.1) with $\boldsymbol{n}_i$, and solving for t results in

$$t = -\frac{\boldsymbol{x}_0 \cdot \boldsymbol{n}_i}{\boldsymbol{v}_0 \cdot \boldsymbol{n}_i}. \tag{A.2}$$

Our interest lies in identifying an optimal direction $\boldsymbol{v}_0$ that minimizes the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_0$.

Proposition 1. *Suppose $\boldsymbol{x}_0 \in P_0$ is fixed and the unit vector $\boldsymbol{v}_0$ can vary while passing through $\boldsymbol{x}_0$ and remaining tangent to P_0. Under the condition that P_0 and P_i are non-parallel, the distance $\|\boldsymbol{x}_i - \boldsymbol{x}_0\|_2$ is minimized if*

$$\boldsymbol{v}_0 = \pm\frac{\alpha\boldsymbol{n}_0 - \boldsymbol{n}_i}{\sqrt{1-\alpha^2}}, \tag{A.3}$$

where $\alpha := \boldsymbol{n}_i \cdot \boldsymbol{n}_0$.

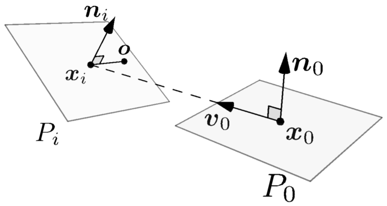

Figure 7: Schematic representation of the ray shooting method from point $\boldsymbol{x}_0$ in direction $\boldsymbol{v}_0$ on hyperplane P_0 to locate point $\boldsymbol{x}_i$ on hyperplane P_i.

Proof. From equation (A.1) and the unit norm of $\boldsymbol{v}_0$, we get $\|\boldsymbol{x}_i - \boldsymbol{x}_0\|_2 = |t|$. Minimizing $|t|$ involves maximizing the absolute value of the denominator in equation (A.2), as the numerator remains constant. The optimization problem is thus defined as

$$\max_{\boldsymbol{v}_0 \in \mathbb{R}^n} |\boldsymbol{v}_0 \cdot \boldsymbol{n}_i|,$$

subject to the constraints

$$\boldsymbol{v}_0 \cdot \boldsymbol{v}_0 = 1 \quad \text{and} \quad \boldsymbol{v}_0 \cdot \boldsymbol{n}_0 = 0,$$

which ensure that $\boldsymbol{v}_0$ remains a unit vector tangent to P_0. Using the method of Lagrange multipliers, we define the Lagrangian function as

$$\mathcal{L}(\boldsymbol{v}_0,\lambda,\mu) := |\boldsymbol{v}_0 \cdot \boldsymbol{n}_i| + \lambda(\boldsymbol{v}_0 \cdot \boldsymbol{v}_0 - 1) + \mu\boldsymbol{v}_0 \cdot \boldsymbol{n}_0,$$

with λ and μ as Lagrangian multipliers. The optimal solution is found at the stationary point of $\mathcal{L}$, where its partial derivatives vanish. Namely,

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{v}_0} = s\boldsymbol{n}_i + 2\lambda\boldsymbol{v}_0 + \mu\boldsymbol{n}_0 = \boldsymbol{0}, \tag{A.4a}$$

$$\frac{\partial\mathcal{L}}{\partial\lambda} = \boldsymbol{v}_0 \cdot \boldsymbol{v}_0 - 1 = 0, \tag{A.4b}$$

$$\frac{\partial\mathcal{L}}{\partial\mu} = \boldsymbol{v}_0 \cdot \boldsymbol{n}_0 = 0, \tag{A.4c}$$

where $s := \operatorname{sgn}(\boldsymbol{v}_0 \cdot \boldsymbol{n}_i)$.

To determine μ, we take the dot product of equation (A.4a) with $\boldsymbol{n}_0$. Incorporating the constraint from equation (A.4c) and recognizing that $\boldsymbol{n}_0 \cdot \boldsymbol{n}_0 = 1$, we deduce that $\mu = -s\boldsymbol{n}_i \cdot \boldsymbol{n}_0 = -s\alpha$. Consequently, resolving equation (A.4a) for $\boldsymbol{v}_0$ leads us to $\boldsymbol{v}_0 = s(\alpha\boldsymbol{n}_0 - \boldsymbol{n}_i)/(2\lambda)$. The next step involves the determination of λ.

We proceed by taking the dot product of equation (A.4a) with $\boldsymbol{v}_0$ and subsequently applying equations (A.4b) and (A.4c), which yields $2\lambda = -s\boldsymbol{n}_i \cdot \boldsymbol{v}_0$. On the other hand, we also take the dot product of equation (A.4a) with $s\boldsymbol{n}_i$, considering the fact that $\boldsymbol{n}_i \cdot \boldsymbol{n}_i = 1$ and $s^2 = 1$. This operation results in the equation $1 + 2\lambda(s\boldsymbol{n}_i \cdot \boldsymbol{v}_0) - \alpha^2 = 0$. By eliminating $s\boldsymbol{n}_i \cdot \boldsymbol{v}_0$ from these two equations and solving for 2λ, we find that $2\lambda = \pm\sqrt{1-\alpha^2}$. Hence, we derive $\boldsymbol{v}_0 = \pm s(\alpha\boldsymbol{n}_0 - \boldsymbol{n}_i)/\sqrt{1-\alpha^2}$. Given the arbitrary nature of the sign in this expression, we can equate $\pm s$ to ± 1, thereby completing the proof. □

Remark 1. When P_0 and P_i are parallel, equation (A.3) becomes undefined, as $\alpha = \pm 1$. In this case, $t = 0$, leading to the trivial solution $\boldsymbol{x}_i = \boldsymbol{x}_0$.